goTenna

Intro to the goTenna SDK

Before the arrival of goTenna technology there was no easily accessible solution for developers who wanted to build applications which required off-grid connectivity at long distances. The only things available were Bluetooth and WiFi that were extremely limited in range. The goal of the goTenna SDK is to expand the toolbox of the development community so that they may build new creative applications that were not possible before. We like to think of ourselves as the "inverted Bluetooth" wherein we offer long-range but relatively lower data throughput than the BT/WiFi options.

We hope that although the goTenna app focuses on chat and GPS sharing, the developer community will come up with unique applications of our capabilities that we may never have considered before!

Primary Functional Features

- Open broadcast messaging
- 1-to-1 individual messaging
- Group messaging
- Completely free-form agnostic data bucket for you to do anything you want, with 2 simple fair-use rules:
    - 256 byte maximum payload size for all transmission types
    - 5 total transmissions allowed per end-user per minute

Other things the SDK will do for you

- Manage Bluetooth connectivity for iOS & Android
- Report on receipt or failure for 1-to-1 messaging
- Option to activate goTenna encryption within 1-to-1 and group messaging
- Report on goTenna battery life and firmware version
- Manage firmware upgrades
- Provide alerts and other diagnostic data as needed
- Maintains separation from other SDK users. To illustrate, App X can only speak to App X on goTenna via a tokening process, apps cannot be interoperable for security reasons. You will need to contact goTenna support for your app token.

Why is the SDK the way it is?

Most of you might wonder why the payload and bandwidth limits are in place. This is purely because spectrum is a finite resource, and the FCC has only made less than 100Khz available where we are operating, which is a miniscule amount of spectrum. For this reason, we all have to be cognizant of our network footprint and put in place some controls so that we don't ruin the airwaves for other SDK users. If we allowed non-stop transmissions, not only would it likely destroy the hardware, but instead of us being able to support a bunch of people in an area, we would only be able to support 1 or 2. So we ask that everyone please be mindful of how you decide to use goTenna. Building an app that just does HD video transfer across a city for fun may technically work if you're patient enough, but it would ruin the experience for anyone else in the area. So play nice please!

goTenna

Development Design Tips

- Keep in mind the purpose of goTenna, which is enabling short-form burst data at long distances. Although larger things can be done with the SDK, it is not what the system is designed for. Using the goTenna for applications which require large throughput will result in lower battery life, less bandwidth for others, latency, and generally just not work well. Plus, it is not nice to others.
- Serialize your data as often as you can. If you have certain pre-set types of information you want to send, you're much better off sending a small burst that represents that data as opposed to the raw data itself. For example, need to have an "Alert = Arriving" or "Alert = Emergency" in your app? Instead of spelling everything out, consider a system of flags and structured data so that you can send something like, "A:1" or "A:2" and then interpret it on the other end!
- Find any opportunity to cut down on your own payload's structural overhead.
- Pre-cache data within your applications whenever possible. The goTenna is best when used to send beacons of information like changes, alerts, or other small bursts that can then be contextualized richly using information that was previously available on the other user's app.
- Keep in mind that Bluetooth Low-Energy has a very low data throughput, so it at times it may be a bottleneck in your communications back and forth from goTenna. Delays may sometimes be from this root cause.
- Try to avoid any communications architectures where you create a situation where many nodes are trying to communicate at the exact same time. For example, a poor architecture would be one where 1 unit sends out a polling message, which then results in every other unit in the area trying to respond simultaneously in some way, this can result in latency and lost packets. A better architecture would be one where units are temporally spread out when trying to communicate to a single unit.

Other Rules:

- SDK terms of use are included in the repository, please read thoroughly.
- It is the responsibility of the developer to comply with all FCC regulations. This includes a ban on store-and-forward operations as well as the transmission of images over MURS frequencies (which is where goTenna operates)
- goTenna reserves the right to revoke SDK access to any application that is found to be abusive and against the spirit of what this tool is designed to be used for, example of such violations include:
    - Patterns of continuous transmissions (large file transfers or any other operations which consistently max out the limits of the SDK on bandwidth use)
    - Excessive automated beaconing which noticeably degrades other goTenna users quality of service
    - Applications violating FCC rules
    - Any evidence/attempts to circumvent the fair-use controls within the SDK protecting public bandwidth